PERCIVAL: Making In-Browser Perceptual Ad Blocking Practical with Deep Learning

Zainul Abi Din, *UC Davis;* Panagiotis Tigas, *University of Oxford;*
Samuel T. King, *UC Davis, Bouncer Technologies;* Benjamin Livshits,
Brave Software, Imperial College London

https://www.usenix.org/conference/atc20/presentation/din

This paper is included in the Proceedings of the
2020 USENIX Annual Technical Conference.

July 15–17, 2020

978-1-939133-14-4

Open access to the Proceedings of the
2020 USENIX Annual Technical Conference
is sponsored by USENIX.

PERCIVAL: Making In-Browser Perceptual Ad Blocking Practical with Deep Learning

Zainul Abi Din [†] *UC Davis*

Panagiotis Tigas[†] *University of Oxford*

Samuel T. King *UC Davis* *Bouncer Technologies*

Benjamin Livshits *Brave Software* *Imperial College London*

Abstract

In this paper we present PERCIVAL, a browser-embedded, lightweight, deep learning-powered ad blocker. PERCIVAL embeds itself within the browser's image rendering pipeline, which makes it possible to intercept every image obtained during page execution and to perform image classification based blocking to flag potential ads.

Our implementation inside both Chromium and Brave browsers shows only a minor rendering performance overhead of 4.55%, for Chromium, and 19.07%, for Brave browser, demonstrating the feasibility of deploying traditionally heavy models (i.e. deep neural networks) inside the critical path of the rendering engine of a browser. We show that our image-based ad blocker can replicate EasyList rules with an accuracy of 96.76%. Additionally, PERCIVAL does surprisingly well on ads in languages other than English and also performs well on blocking first-party Facebook ads, which have presented issues for rule-based ad blockers. PERCIVAL proves that image-based perceptual ad blocking is an attractive complement to today's dominant approach of block lists.

1 Introduction

Web advertising provides the financial incentives necessary to support most of the free content online, but it comes at a security and privacy cost. To make advertising effective, ad networks or publishers track user browsing behavior across multiple sites to generate elaborate user profiles for targeted advertising.

Users find that ads are intrusive [61] and cause disruptive browsing experience [6, 27]. In addition, studies have shown that advertisements impose privacy and performance costs to users, and carry the potential to be a malware delivery vector [2, 35, 37, 54, 55, 76].

Ad blocking is a software capability for filtering out unwanted advertisements to improve user experience, performance, security, and privacy. At present, ad blockers either run directly in the browser [4, 12] or as browser extensions [1].

Current ad blocking solutions filter undesired content based on "handcrafted" filter lists such as EasyList [74], which contain rules matching ad-carrying URLs and DOM elements. Most widely-used ad blockers, such as uBlock Origin [26] and Adblock Plus [1] use these block lists for content blocking. While useful, these approaches fail against adversaries who can change the ad-serving domain or obfuscate the web page code and metadata.

In an attempt to find a more flexible solution, researchers have proposed alternative approaches to ad blocking. One such approach is called Perceptual ad blocking, which relies on "visual cues" frequently associated with ads like the AdChoices logo or a sponsored content link. Storey et al. [70] built the first perceptual ad blocker that uses traditional computer vision techniques to detect ad-identifiers. Recently, Adblock Plus developers built filters into their ad blocker [15] to match images against a fixed template in order to detect ad labels. Due to the plethora of ad-disclosures, AdChoices logo and other ad-identifiers, it is unlikely that traditional computer vision techniques are sufficient and generalizable to the range of ads one is likely to see in the wild.

A natural extension to traditional vision-based blocking techniques is deep learning. Adblock Plus recently proposed SENTINEL [65] that detects ads in web pages using deep learning. SENTINEL's deep learning model takes as input the screenshot of the rendered webpage to detect ads. However, this technology is still in development.

To this end, we present PERCIVAL, a native, deep learning-powered *perceptual ad blocker*, which is built into the browser image rendering pipeline. PERCIVAL intercepts every image obtained during the execution sequence of a page and blocks images that it classifies as ads. PERCIVAL is small (half the average webpage size [25]) and fast, and we deploy it online within two commercial browsers to block and detect ads at real-time.

PERCIVAL can be run *in addition* to an existing ad blocker, as a last-step measure to block whatever slips through its

[†]Employed by Brave software when part of this work took place.

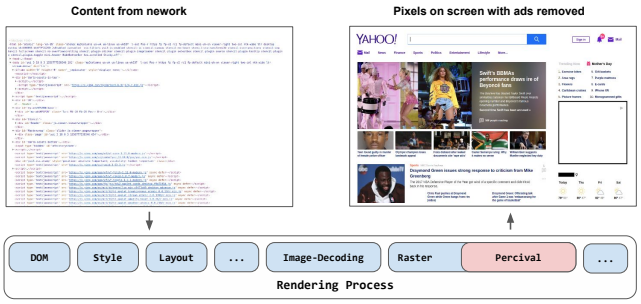


Figure 1: Overall architecture of PERCIVAL. PERCIVAL is positioned in the renderer process-which is responsible for creating rasterized pixels from HTML, CSS, Javascript. As the renderer process creates the DOM and decodes and rasterizes all image frames, these are first passed through PERCIVAL. PERCIVAL blocks the frames that are classified as ads. The corresponding output with ads removed is shown above (right).

filters. However, PERCIVAL may also be deployed *outside* the browser, for example, as part of a crawler, whose job is to construct comprehensive block lists to supplement EasyList.

1.1 Contributions

This paper makes the following contributions:

- **Perceptual ad blocking in Chromium-based browsers.** We deploy PERCIVAL in two Chromium-based browsers: Chromium and Brave. We demonstrate two deployment scenarios; first, PERCIVAL blocks ads synchronously as it renders the page, with a modest performance overhead. Second, PERCIVAL classifies images *asynchronously* and memoizes the results, thus speeding up the classification process[1].
- **Lightweight and accurate deep learning models.** We show that ad blocking can be done effectively using highly-optimized deep neural network-based models for image processing. Previous studies suggest that models over 5MB in size become hard to deploy on mobile devices [62]; because of our focus on low-latency detection, we create a compressed in-browser model that occupies 1.76MB[2] on disk, which is smaller by factor of 150 compared to other models of this kind [22], while maintaining similar accuracy results.
- **Accuracy and performance overhead measurements.** We show that our perceptual ad blocking model can replicate EasyList rules with the accuracy of 96.76%, making PERCIVAL a viable and complementary ad blocking layer. Our implementation within Chromium shows an average overhead of 178.23ms for page rendering. This overhead shows the feasibility of deploying deep neural networks inside the critical path of the rendering engine of the browser.
- **First-party ad blocking.** While the focus of traditional ad blocking is primarily on third-party ad blocking, we show that PERCIVAL blocks first-party ads as well, such as those found on Facebook. Specifically, our experiments show that PERCIVAL blocks ads on Facebook (often referred to as "sponsored content") with a 92% accuracy, with precision and recall of 78.4% and 70.0%.
- **Language-agnostic blocking.** We demonstrate that our model in PERCIVAL blocks images that are in languages we did not train our model on. We evaluate our trained model on Arabic, Chinese, Korean, French and Spanish image-based ads. Our model achieves an accuracy of 81.3% on Arabic, 95.1% on Spanish, and 93.9% on French datasets, with moderately high precision and recall. However, results from Chinese and Korean ads are less accurate.

[1] We make the source code, pre-trained models and data available for other researchers at https://github.com/dxaen/percival

[2] Our in-browser model is 3.2MB due to a less efficient serialization format. Still, the weights are identical to our 1.76MB model

2 Motivation

Intrusive, online, advertising has been a long standing concern for user privacy, security and overall web experience. While web advertising makes it easier and more economic for businesses to reach a wider audience, bad actors have exploited this channel to engage in malicious activities. Attackers use ad-distribution channels to hijack compromised web pages in order to trick users into downloading malware [54]. This is known as malicious advertising.

Mobile users are also becoming targets of malicious advertising [68]. Mobile applications contain code embedded from the ad networks, which provides the interface for the ad networks to serve ads. This capability has been abused by attackers where the landing page of the advertisements coming from ad networks links to malicious content. Moreover, intrusive advertisements significantly affect the user experience on mobile phones due to limited screen size [38]. Mobile ads also drain significant energy and network data [73].

Web advertising also has severe privacy implications for users. Advertisers use third party web-tracking by embedding code in the websites the users visit, to identify the same users again in a different domain, creating a more global view of the user browsing behavior [52]. Private user information is collected, stored and sold to other third party advertisers. These elaborate user profiles can be used to infer sensitive information about the users like medical history or political views [31,57]. Communication with these third party services is unencrypted, which can be exploited by attackers.

The security and privacy concerns surrounding web advertising has motivated research in ad blocking tools

from both academia [29,40,44,69,75] and industry notably Adblock Plus [1], Ghostery [13], Brave [4], Mozilla [47], Opera [16] and Apple [17]. Ad blocking serves to improve web security, privacy, usability, and performance. As of February 2017, 615 million devices had ad blockers installed [19] However, recently Google Chrome [14] and Safari [3] proposed changes in the API exposed to extensions, with the potential to block extension based ad-blockers. This motivates the need for native ad blockers like Brave [4], Opera [16], AdGraph [44], PageGraph [32] and even PERCIVAL.

3 PERCIVAL Overview

This paper presents PERCIVAL, a novel deep-learning based system for blocking ads. Our primary goal is to build a system that blocks ad images that could escape detection by current techniques, while remaining small and efficient enough to run in a mobile browser.

Figure 1 shows how PERCIVAL blocks rendering of ads. First, PERCIVAL runs in the browser image rendering pipeline. By running in the image rendering pipeline, PERCIVAL can inspect all images before the browser shows them to the user. Second, PERCIVAL uses a deep convolutional neural network (CNN) for detecting ad images. Using CNNs enables PERCIVAL to detect a wide range of ad images, even if they are in a language that PERCIVAL was not trained on.

This section discusses PERCIVAL's architecture overview, possible alternative implementations and detection model. Section 4 discusses the detailed design and implementation for our browser modifications and our detection model.

3.1 PERCIVAL's Architecture Overview

PERCIVAL's detection module runs in the browser's image decoding pipeline after the browser has decoded the image into pixels, but before it displays these pixels to the user. Running PERCIVAL after the browser has decoded an image takes advantage of the browser's mature, efficient, and extensive image decoding logic, while still running at a choke point before the browser displays the decoded pixels. Simply put, if a user sees an image, it goes through this pipeline first.

More concretely, as shown in Figure 1 PERCIVAL runs in the renderer process of the browser engine. The renderer process on receiving the content of the web page proceeds to create the intermediate data structures to represent the web page. These intermediate representations include the DOM-which encodes the hierarchical structure of the web page, the layout-tree, which consists of the layout information of all the elements of the web page, and the display list, which includes commands to draw the elements on the screen. If an element has an image contained within it, it needs to go through the *Image Decoding Step* before it can be rasterized. We run PERCIVAL after the *Image Decoding Step* during the *raster* phase which helps run PERCIVAL in parallel for multiple images at a time. Images that are classified as ads

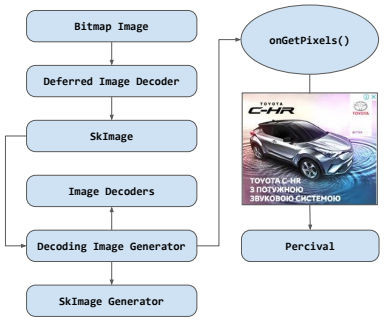


Figure 2: PERCIVAL in the image decoding pipeline. SkImage Generator allocates a bitmap and calls the `onGetPixels()` of `DecodingImageGenerator` to populate the bitmap. This bitmap is then passed to the network for classification and cleared if it contains an ad.

are blocked from rendering. The web page with ads removed is shown in Figure 1 (right). We present the detailed design and implementation in Section 4.

3.2 Alternative Possible Implementations and Advantages of PERCIVAL

One alternative to running PERCIVAL directly in the browser could have been to run PERCIVAL in the browser's JavaScript layer via an extension. However, this would require scanning the DOM to find image elements, waiting for them to finish loading, and then screenshotting the pixels to run the detection model. The advantage of a JavaScript-based system is that it works within current browser extensibility mechanisms, but recent work has shown how attackers can evade this style of detection [71].

Ad blockers that inspect web pages based on the DOM such as Ad Highlighter [70] are prone to DOM obfuscation attacks. They assume that the elements of the DOM strictly correspond to their visual representation. For instance, an ad blocker that retrieves all `img` tags and classifies the content contained in these elements does not consider the case, where a rendered image is a result of several CSS or JavaScript transformations and not the source contained in the tag. These ad blockers are also prone to resource exhaustion attacks where the publisher injects a lot of dummy elements in the DOM to overwhelm the ad blocker.

Additionally, a native implementation is much faster than a browser extension implementation with the added benefit of having access to the unmodified image buffers.

3.3 Detection Model

PERCIVAL runs a detection model on every image loaded in the document's main frame, a sub-document such as an `iframe`, as well as images loaded in JavaScript to determine if the image is an ad.

Although running directly within the browser provides PERCIVAL with more control over the image rendering

process, it introduces a challenge: how to run the model efficiently in a browser? Our goal is to run PERCIVAL in browsers that run on laptops or even mobile phones. This requires that the model be small to be practical [62]. This design also requires that the model run directly in the image rendering pipeline, so overhead remains low. Any overhead adds latency to rendering for all images it inspects.

In PERCIVAL, we use the SqueezeNet [43] CNN as the starting point for our detection model. We modify the basic SqueezeNet network to be optimized for ad blocking by removing less important layers. This results in a model size that is less than 2MB and detects ad images in 11ms per image.

A second challenge in using small CNNs is how to provide enough training data. In general, smaller CNNs can have suitable performance but require more training data. What is more, the labels are highly imbalanced making the training procedure even more challenging.

Gathering ad images is non-trivial; most ads are programmatically inserted into the document through `iframes` or JavaScript, and so simple crawling methods that work only on the initial HTML of the document will miss most of the ad images.

To crawl ad images, other researchers [22, 71] propose screenshotting iframes or JavaScript elements. This data collection method leads to problems with synchronizing the timing of the screenshot and when the element loads. Many screenshots end up with whites-space instead of the image content. Also, this method only makes sense if the input to the classifier is the rendered content of the web page.

To address these concerns and to provide ample training data, we design and implement a custom crawler in Blink[3] that handles dynamically-updated data and eliminates the race condition between the browser displaying the content and the screenshot we use to capture the image data. Our custom-crawler fetches ad and non-ad images directly from the rendering pipeline and uses the model trained during the previous phase as a labeler. This way we amplify our dataset to fine-tune our model further.

4 Design and Implementation of PERCIVAL

This section covers the design and implementation of the browser portion of PERCIVAL. We first cover the high-level design principles that guide our design, and then we discuss rendering and image handling in Blink, the rendering engine of Chromium-based browsers. Finally, we describe our end-to-end implementation within Blink.

4.1 Design Goals

We have two main goals in our design of PERCIVAL:

Run PERCIVAL at a choke point: Advertisers can serve ad images in different formats, such as JPG, PNG, or GIF. Depending on the format of the image, an encoded frame can traverse different paths in the rendering pipeline. Also, a wide range of web constructs can cause the browser to load images, including HTML image tags, JavaScript image objects, HTML Canvas elements, or CSS background attributes. Our goal is to find a single point in the browser to run PERCIVAL, such that it inspects all images, operates on pixels instead of encoded images, but does so before the user sees the pixels on the screen, enabling PERCIVAL to block ad images cleanly. **Note:** If individual pixels are drawn programmatically on canvas, PERCIVAL will not block it from rendering.

In Blink, the raster task within the rendering pipeline enables PERCIVAL to inspect, and potentially block, all images. Regardless of the image format or how the browser loads it, the raster task decodes the given image into raw pixels, which it then passes to the GPU to display the content on the screen. We run PERCIVAL at this precise point to abstract different image formats and loading techniques, while still retaining the opportunity to block an image before the user sees it.

Run multiple instances of PERCIVAL in parallel: Running PERCIVAL in parallel is a natural design choice because PERCIVAL makes all image classification decisions independently based solely on the pixels of each individual image. When designing PERCIVAL, we look for opportunities to exploit this natural parallelism to minimize the latency added due to the addition of our ad blocking model.

4.2 Rendering and PERCIVAL: Overview

We integrate PERCIVAL into Blink, the rendering engine for Google Chrome and Brave. From a high level, Blink's primary function is to turn a web page into the appropriate GPU calls [5] to show the user the rendered content.

A web page can be thought of as a collection of HTML, CSS, and JavaScript code, which the browser fetches from the network. The rendering engine parses this code to build the DOM and layout tree, and to issue OpenGL calls via `Skia`, Google's graphics library [24].

The layout tree contains the locations of the regions the DOM elements will occupy on the screen. This information together with the DOM element is encoded as a `display item`.

The browser then proceeds with rasterization, which takes the display items and turns them into bitmaps. Rasterization issues OpenGL draw calls via the Skia library to draw bitmaps. If the display list items have images in them (a common occurrence), the browser must decode these images before drawing them via Skia.

PERCIVAL intercepts the rendering process at this point, after the `Image Decode Task` and during the `Raster Task`. As the renderer process creates the DOM and decodes and rasterizes all image frames, these are first passed through

[3] Blink `http://www.chromium.org/blink` is the rendering engine used by Chromium.

PERCIVAL. PERCIVAL blocks the frames that are classified as ads.

4.3 End-to-End Implementation in Blink

We implement PERCIVAL inside Blink (Chromium rendering engine), where PERCIVAL uses the functionality exposed by the Skia library. Skia uses a set of image decoding operations to turn `SkImages`, which is the internal type within Skia that encapsulates images, into bitmaps. PERCIVAL reads these bitmaps and classifies their content accordingly. If PERCIVAL classifies the bitmap as an ad, it blocks it by removing its content. Otherwise, PERCIVAL lets it pass through to the next layers of the rendering process. When content is cleared, there are several ways to fill up the surrounding white-space; either collapsing it by propagating the information upwards or displaying a predefined image (user's spirit animal) in place of the ad.

Figure 2 shows an overview of our Blink integration. Blink class `BitmapImage` creates an instance of `DeferredImageDecoder` which in turn instantiates a `SkImage` object for each encoded image. SkImage creates an instance of `DecodingImageGenerator` (blink class) which will in turn decode the image using the relevant image decoder from Blink. Note that the image hasn't been decoded yet since chromium practices deferred image decoding.

Finally, `SkImageGenerator` allocates bitmaps corresponding to the encoded `SkImage`, and calls `onGetPixels()` of `DecodingImageGenerator` to decode the image data using the proper image decoder. This method populates the buffer (pixels) that contain decoded pixels, which we pass to PERCIVAL along with the image height, width, channels information (`SKImageInfo`) and other image metadata. PERCIVAL reads the image, scales it to $224 \times 224 \times 4$ (default input size expected by SqueezeNet), creates a tensor, and passes it through the CNN. If PERCIVAL determines that the buffer contains an ad, it clears the buffer, effectively blocking the image frame.

Rasterization, image decoding, and the rest of the processing happen on a raster thread. Blink rasters on a per tile basis and each tile is like a resource that can be used by the GPU. In a typical scenario there are multiple raster threads each rasterizing different raster tasks in parallel. PERCIVAL runs in each of these worker threads after image decoding and during rasterization, which runs the model in parallel.

As opposed to Sentinel [65] and Ad Highlighter [36] the input to PERCIVAL is not the rendered version of web content; PERCIVAL takes in the Image pixels directly from the image decoding pipeline. This is important since with PERCIVAL we have access to unmodified image buffers and it helps prevent attacks where publishers modify content of the webpage (including iframes) with overlaid masks (using CSS techniques) meant to fool the ad blocker classifier.

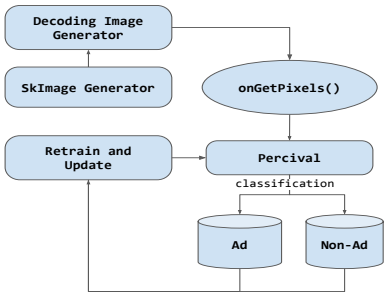


Figure 3: Crawling, labelling and re-training with PERCIVAL. Every decoded image frame is passed through PERCIVAL and PERCIVAL downloads the image frame into the appropriate bucket.

5 Deep Learning Pipeline

This section covers the design of PERCIVAL's deep neural network and the corresponding training workflow. We first describe the network employed by PERCIVAL and the training process. We then describe our data acquisition and labelling techniques.

5.1 PERCIVAL's CNN Architecture

We cast ad detection as a traditional image classification problem, where we feed images into our model and it classifies them as either being (1) an ad, or (2) not an ad. CNNs are the current standard in the computer vision community for classifying images.

Because of the prohibitive size and speed of standard CNN based image classifiers, we use a small network, SqueezeNet [43], as the starting point for our in-browser model. The SqueezeNet authors show that SqueezeNet achieves comparable accuracy to much larger CNNs, like AlexNet [48], and boasts a final model size of 4.8 MB.

SqueezeNet consists of multiple *fire modules*. A *fire module* consists of a "squeeze" layer, which is a convolution layer with 1×1 filters and two "expand" convolution layers with filter sizes of 1×1 and 3×3, respectively. Overall, the "squeeze" layer reduces the number of input channels to larger convolution filters in the pipeline.

A visual summary of PERCIVAL's network structure is shown in Figure 4. As opposed to the original SqueezeNet, we down-sample the feature maps at regular intervals in the network. This helps reduce the classification time per image. We also perform max-pooling after the first convolution layer and after every two fire modules.

5.2 Data Acquisition

We use two systems to collect training image data. First, we use a traditional crawler with traditional ad-blocking rules (EasyList [7]) to identify ad images. Second, we use our browser instrumentation from PERCIVAL to collect images, improving on some of the issues we encountered with our traditional crawler.

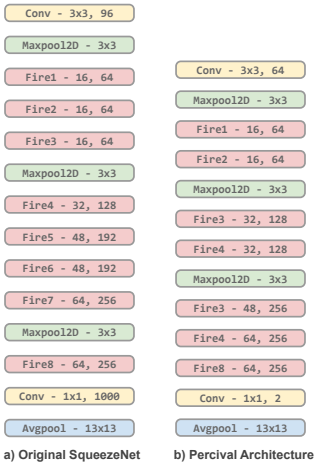


Figure 4: Original SqueezeNet (left) and PERCIVAL's fork of SqueezeNet (right). For `Conv`, `Maxpool2D`, and `Avgpool` blocks $a \times b$ represents the dimensions of the filters used. For fire blocks *a, b* represents the number of intermediate and output channels. We remove extraneous blocks as well as downsample the feature maps at regular intervals to reduce the classification time per image.

5.2.1 Crawling with EasyList

We use a traditional crawler matched with a traditional rule-based ad blocker to identify ad content for our first dataset. In particular, to identify ad elements which could be iframes or complex JavaScript constructs, we use EasyList, which is a set of rules that identify ads based on the URL of the elements, location within the page, origin, class or id tag, and other hand-crafted characteristics known to indicate the presence of ad content.

We built a crawler using Selenium [21] for browser automation. We then use the crawler to visit Alexa top-1,000 web sites, waiting for 5 seconds on each page, and then randomly selecting 3 links and visiting them, while waiting on each page for a period of 5 seconds as before. For every visit, the crawler applies every EasyList network, CSS and exception rule.

For every element that matches an EasyList rule, our crawler takes a screenshot of the component, cropped tightly to the coordinates reported by Chromium, and then stores it as an ad sample. We capture non-ad samples by taking screenshots of the elements that do *not* match any of the EasyList rules. Using this approach we, extract 22,670 images out of which 13,741 are labelled as ads, and 8,929 as non-ads. This automatic process was followed by a semi-automated post-processing step, which includes removing duplicate images, as well as manual spot-checking for misclassified images.

Eventually, we identify 2,003 ad images and 7,432 non-ad images. The drop in the number of ad images from 13,741 to 2,003 is due to a lot duplicates and content-less (single-color) images due to the asynchrony of iframe-loading and the timing of the screenshot. These shortcomings motivated our new crawler. To balance the positive and negative examples in our dataset so the classifier doesn't favor one class over another, we limited the number of non ad and ad images to 2,000.

5.2.2 Crawling with PERCIVAL

We found that traditional crawling was good enough to bootstrap the ad classification training process, but it has the fundamental disadvantage that for dynamically-updated elements, the meaningful content is often unavailable at the time of the screenshot, leading to screenshots filled with white-space.

More concretely, the *page load* event is not very reliable when it comes to loading iframes. Oftentimes when we take a screenshot of the webpage after the *page load* event, most of the iframes do not appear in the screenshots. Even if we wait a fixed amount of time before taking the screenshot, iframes constantly keep on refreshing, making it difficult to capture the rendered content within the iframe consistently.

To handle dynamically-updated data, we use PERCIVAL's browser architecture to read all image frames after the browser has decoded them, eliminating the race condition between the browser displaying the content and the screenshot we use to capture the image data. This way we are guaranteed to capture all the iframes that were rendered, independently of the time of rendering or refresh rate.

Instrumentation: Figure 3 shows how we use PERCIVAL's browser instrumentation to capture image data. Each encoded image invokes an instance of `DecodingImageGenerator` inside Blink, which in turn decodes the image using the relevant image decoder (PNG, GIFs, JPG, etc.). We use the buffer passed to the decoder to store pixels in a bitmap image file, which contains exactly what the rendering engine sees. Additionally, the browser passes this decoded image to PERCIVAL, which determines whether the image contains an ad. This way, every time the browser renders an image, we automatically store it and label it using our initially trained network, resulting in a much cleaner dataset.

Crawling: To crawl for ad and non-ad images, we run our PERCIVAL-based crawler with a browser automation tool called Puppeteer [20]. In each phase, the crawler visits the landing page of each Alexa top-1,000 websites, waits until `networkidle0` (when there are no more than 0 network connections for at least 500 ms) or 60 seconds. We do this to ensure that we give the ads enough time to load. Then our crawler finds all internal links embedded in the page. Afterwards, it visits 20 randomly selected links for each page, while waiting for `networkidle0` event or 60 seconds time out on each request.

In each phase, we crawl between 40,000 to 60,000 ad images. We then post process the images to remove duplicates, leaving around 15-20% of the collected results as useful. We

Images	Ads Identified	Accuracy	Precision	Recall
6,930	3466	96.76%	97.76%	95.72%

Figure 5: Summary of the results obtained by testing the dataset gathered using EasyList with PERCIVAL.

crawl for a total of 8 phases, retraining PERCIVAL after each stage with the data obtained from the current and all the previous crawls. As before, we cap the number of non-ad images to the amount of ad images to ensure a balanced dataset.

This process was spread-out in time over 4 months, repeated every 15 days for a total of 8 phases, where each phase took 5 days. Our final dataset contains 63,000 unique images in total with a balanced split between positive and negative samples.

6 Evaluation

6.1 Accuracy Against EasyList

To evaluate whether PERCIVAL can be a viable shield against ads, we conduct a comparison against the most popular crowd-sourced ad blocking list, EasyList [7], currently being used by extensions such as Adblock Plus [1], uBlock Origin [26] and Ghostery [13].

Methodology: For this experiment, we crawl Alexa top 500 news websites as opposed to Alexa top 1000 websites used in the crawl for training. This is because news websites are an excellent source of advertisements [18] and the crawl can be completed relatively quickly. Also, Alexa top 500 news websites serves as a test domain different from the train domain we used previously.

For our comparison we create two data sets: First, we apply EasyList rules to select DOM elements that potentially contain ads (IFRAMEs, DIVs, etc.); we then capture screenshots of the contents of these elements. Second, we use resource-blocking rules from EasyList to label all the images of each page according to their resource URL. After crawling, we manually label the images to identify the false positives resulting in a total of 6,930 images.

Performance: On our evaluation dataset, PERCIVAL is able to replicate the EasyList rules with accuracy 96.76%, precision 97.76% and recall 95.72% (Figure 5), illustrating a viable alternative to the manually-curated filter-lists.

Ads	No-ads	Accuracy	FP	FN	Precision	Recall
354	1,830	92.0%	68	106	78.4%	70.0%

Figure 6: Online evaluation of Facebook ads and sponsored content.

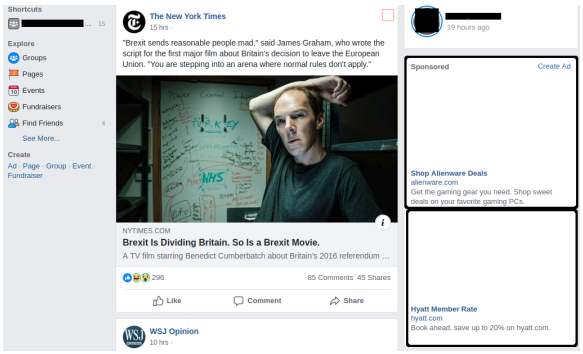


Figure 7: The screenshots show one of the author' Facebook home page accessed with PERCIVAL. The black rectangles are not part of the original screenshot.

6.2 Blocking Facebook Ads

Facebook obfuscates the "signatures" of ad elements (e.g. HTML classes and identifiers) used by filter lists to block ads since its business model depends on serving first-party ads. As of now, Facebook does not obfuscate the content of sponsored posts and ads due to the regulations regarding misleading advertising [10, 11]. Even though this requirement favors perceptual ad blockers over traditional ones, a lot of the content on Facebook is user-created which complicates the ability to model ad and non-ad content.

In this section, we assess the accuracy of PERCIVAL on blocking Facebook ads and sponsored content.

Methodology: To evaluate PERCIVAL's performance on Facebook, we browse Facebook with PERCIVAL for a period of 35 days using two non-burner accounts that have been in use for over 9 years. Every visit is a typical Facebook browsing session, where we browse through the feed, visit friends' profiles, and different pages of interest. For desktop computers two most popular places to serve ads is the right-side columns and within the feed (labelled sponsored) [9].

For our purposes, we consider content served in these elements as ad content and everything else as non-ad content. A false positive (FP) is defined as the number of non-ads incorrectly blocked and false negative (FN) is the number of ads PERCIVAL missed to block. For every session, we manually compute these numbers. Figure 6 shows the aggregate numbers from all the browsing sessions undertaken. Figure 7 shows PERCIVAL blocking right-side columns correctly.

Results: Our experiments show that PERCIVAL blocks ads on Facebook with a 92% accuracy and 78.4% and 70.0% as precision and recall, respectively. Figure 6 shows the complete results from this experiment. Even though we achieve the accuracy of 92%, there is a considerable number of false positives and false negatives, and as such, precision and recall are lower. The classifier always picks out the ads in the

right-columns but struggles with the ads embedded in the feed. This is the source of majority of the false negatives. False positives come from high "ad intent" user-created content, as well as content created by brand or product pages on Facebook (Figure 8).

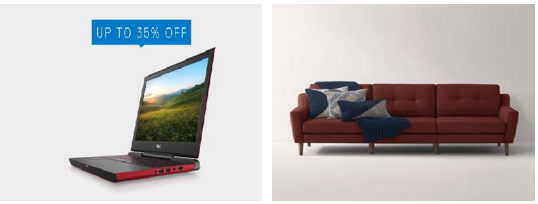


Figure 8: Examples of false positives and false negatives on Facebook (left) **False Positive:** This post was created by page owned by Dell Corp. (right) **False Negative:** This post was part of the sponsored content in the news feed.

Discussion: False Positives and False Negatives: To put Figure 6 into perspective since it might appear to have an alarming number of false positives and false negatives, it is worthwhile to consider an average scenario. If each facebook visit on average consists of browsing through 100 images, then by our experiments, a user will find roughly 16 ad images and 84 non-ad images, out of which PERCIVAL will block 11 to 12 ad images on average while also blocking 3 to 4 non-ad images. This is shown in Figure 10.

In addition to the above mentioned experiments which evaluate the out of box results of using PERCIVAL, we trained a version of PERCIVAL on a particular user's ad images. The model achieved higher precision and recall of 97.25%, 88.05% respectively.

6.3 Blocking Google Image Search Results

To improve our understanding of the misclassifications of PERCIVAL, we used Google Images as a way to fetch images from distributions that have high or low ad intent. For example, we fetched results with the query "Advertisement" and used PERCIVAL to classify and block images. As we can see in Figure 11, out of the top 23 images, 20 of them were successfully blocked. Additionally, we tested with examples of low ad intent distribution we used the query "Obama"). We also searched for other keywords, such as "Coffee", "Detergent", etc. The detailed results are presented in Figure 12. As shown, PERCIVAL can identify a significant percentage of images on a highly ad-biased content.

6.4 Language-Agnostic Detection

We test PERCIVAL against images with language content different than the one we trained on. In particular, we source a data set of images in Arabic, Chinese, French, Korean and Spanish.

Crawling: To crawl for ad and non-ad images, we use ExpressVPN [8] to VPN into major world cities where

Language	# crawled	# Ads	Accuracy	Precision	Recall
Arabic	5008	2747	81.3%	83.3%	82.5%
Spanish	2539	309	95.1%	76.8%	88.9%
French	2414	366	93.9%	77.6%	90.4%
Korean	4296	506	76.9%	54.0%	92.0%
Chinese	2094	527	80.4%	74.2%	71.5%

Figure 9: Accuracy of PERCIVAL on ads in non-English languages. The second column represents the number of images we crawled, while the third column is the number of images that were identified as ads by a native speaker. The remaining columns indicate how well PERCIVAL is able to reproduce these labels.

Images	Ads	No-ads	FP	FN
100	16	84	3-4	4-5

Figure 10: Average reporting of evaluation of Facebook ads and sponsored content per visit. We assume each Facebook visit consists of browsing through 100 total images.

the above mentioned languages are spoken. For instance, to crawl Korean ads, we VPN into two locations in Seoul. We then manually visit top 10 websites as mentioned in SimilarWeb [23] list. We engage with the ad-networks by clicking on ads, as well as closing the ads (icon at the top right corner of the ad) and then choosing random responses like content not relevant or ad seen multiple times. This is done to ensure we are served ads from the language of the region.

We then run PERCIVAL-based crawler with the browser automation tool Puppeteer [20]. Our crawler visits the landing page of each top 50 SimilarWeb websites for the given region, waits until `networkidle0` (when there are no more than 0 network connections for at least 500 ms) or 60 seconds. Then our crawler finds all internal links embedded in the page. Afterwards, it visits 10 randomly selected links for each page, while waiting for `networkidle0` event or 60 seconds time out

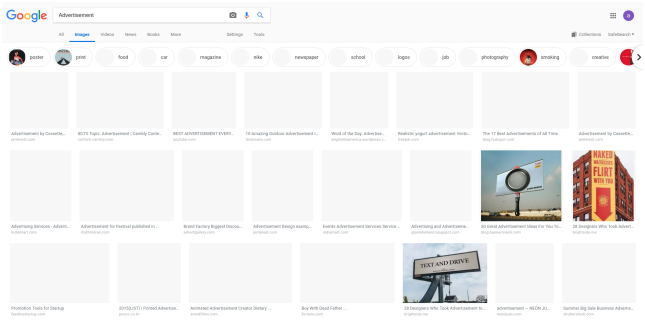


Figure 11: Search results from searching for "Advertisement" on Google images, using PERCIVAL.

Query	# blocked	# rendered	FP	FN
Obama	12	88	12	0
Advertisement	96	4	0	4
Coffee	23	77	-	-
Detergent	85	15	10	6
iPhone	76	24	23	1

Figure 12: PERCIVAL blocking image search results. For each search we only consider the first 100 images returned ("-" represents cases where we were not able to determine whether the content served is ad or non-ad).

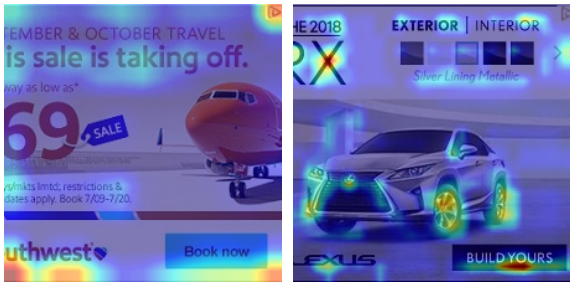
on each request. As opposed to Section 5.2.2, we download every image frame to a single bucket.

Labeling: For each language, we crawl 2,000–6,000 images. We then hire a native speaker of the language under consideration and have them label the data crawled for that language. Afterwards, we test PERCIVAL with this labeled dataset to determine how accurately can PERCIVAL reproduce these human annotated labels. Figure 9 shows the detailed results from all languages we test on. Figure 14 shows a screen shot of a Portuguese website rendered with PERCIVAL.

Results: Our experiments show that PERCIVAL can generalize to different languages with high accuracy (81.3% for Portuguese, 95.1% for Spanish, 93.9% for French) and moderately high precision and recall (83.3%, 82.5% for Arabic, 76.8%, 88.9% for Spanish, 77.6%, 90.4% for French). This illustrates the out-of-the box benefit of using PERCIVAL for languages that have much lower coverage of EasyList rules, compared to the English ones. The model does not perform as well on Korean and Chinese datasets.

6.5 Salience Map of the CNN

To visualize which segments of the image are influencing the classification decision, we used Grad-CAM [64] network salience mapping which allow us to highlight the important regions in the image that caused the prediction. As we can see in Figure 13, our network is focusing on ad visual cues (*AdChoice* logo), when this is present (case (a)), also it follows the outlines of text (signifying existence of text between white space) or identifies features of the object of interest (wheels of a car).



(a) Ad image: Layer 9 (b) Ad image: Layer 5

Figure 13: Salience map of the network on a sample ad images. Each image corresponds to the output of Grad-CAM [64] for the layer in question.

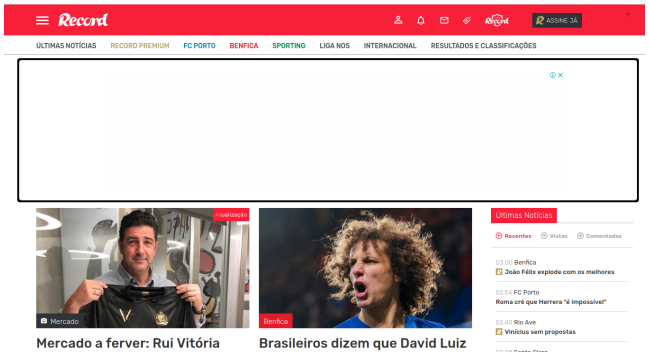


Figure 14: PERCIVAL results on `record.pt` (Portuguese language website).

6.6 Runtime Performance Evaluation

We next evaluate the impact of PERCIVAL-based blocking on the browser performance. This latency is a function to the number and complexity of the images on the page and the time the classifier takes to classify each of them. We measure the rendering time impact when we classify each image *synchronously*.

To evaluate the performance of our system, we used top 5,000 URLs from Alexa to test against Chromium compiled on Ubuntu Linux 16.04, with and without PERCIVAL activated. We also tested PERCIVAL in Brave, a privacy-oriented Chromium-based browser, which blocks ads using block lists by default. For each experiment we measured `render` time which is defined as the difference between `domComplete` and `domLoading` events timestamps. We conducted the evaluations sequentially on the same Amazon m5.large EC2 instance to avoid interference with other processes and make the comparison fair. Also, all the experiments were using `xvfb` for rendering, an in-memory display server which allowed us to run the tests without a display.

In our evaluation we show an increase of 178.23ms of median render time when running PERCIVAL in the rendering critical path of Chromium and 281.85ms when running inside Brave browser with ad blocker and shields on. Figures 15 and 16 summarize the results.

To capture rendering and perceptual impact better, we create a micro-benchmark with `firstMeaningfulPaint` to illustrate overhead. In our new experiment, we construct a static html page containing 100 images. We then measure `firstMeaningfulPaint` with Percival classifying images synchronously and asynchronously. In synchronous classification, PERCIVAL adds 120ms to Chrome and 140ms

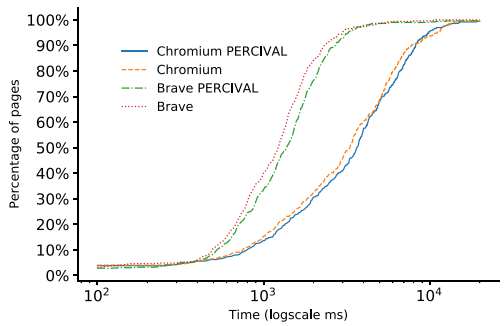


Figure 15: `Render` time evaluation in Chromium and Brave browser.

to Brave. In asynchronous classification, PERCIVAL adds 6ms to Chrome and 3ms to Brave. Although asynchronous classification nearly eliminates overhead, it opens up the possibility of showing an image to the user that we later remove after flagging it as an ad because the rasterization of the image runs in parallel with classification in this mode of operation.

To determine why PERCIVAL with Brave is slower than Chromium. We trace events inside the decoding process using `firstMeaningfulPaint` and confirm there is no significant deviation between the two browsers. The variance observed initially is due to the additional layers in place like Brave's ad blocking shields.

6.7 Comparison With Other Deep Learning Based Ad Blockers

Recently, researchers evaluated the accuracy of three deep-learning based perceptual ad blockers including PERCIVAL [71]. They used real website data from Alexa top 10 news websites to collect data which is later manually labelled. In this evaluation, PERCIVAL outperformed models 150 times bigger than PERCIVAL in terms of recall. We show their results in Figure 17.

6.8 Adversarial Attacks against PERCIVAL

In recent work by Tramèr et al. [71], they show how the implementation of some state-of-the-art perceptual ad blockers, including PERCIVAL, is vulnerable to attacks.

First, the authors in [71] claim that one adversarial sample influences PERCIVAL to block another benign non-ad image. This, however, is not true; the authors claim to use two benign images, one of which is not benign and other is contentless white-space image. PERCIVAL blocks these images. If these are replaced with stock non-ad images, PERCIVAL correctly renders both, meaning that PERCIVAL makes each decision independently and is not vulnerable to hijacking as is claimed in the paper.

Baseline	Treatment	Overhead (%)	(ms)
Chromium	Chromium + PERCIVAL	4.55	178.23
Brave	Brave + PERCIVAL	19.07	281.85

Figure 16: Performance evaluation of PERCIVAL on `Render` metric.

We found that one of the attacks where they used PERCIVAL's model to create adversarial ad images affects PERCIVAL due to our design decision to run PERCIVAL client-side thereby giving attackers white box access to the model. To address this concern, we argue that PERCIVAL is extremely light-weight and can be re-trained and updated very quickly. Our model currently takes 9 minutes (7 epochs) to fine-tune the weights of the network on an NVIDIA V 100 GPU, meaning that we can generate new models very quickly. PERCIVAL is 1.7MB which is almost half the average web page in 2018 [25] making frequent downloads easier.

To demonstrate, re-training and model update as an effective defense against the adversarial samples, we trained a MobilenetV2 [63] with our current dataset. It took 9 minutes of fine-tuning to get to our baseline accuracy. The updated model correctly classified all the adversarial samples generated for PERCIVAL by Tramer et al. [71] suggesting that none of the samples transferred to this model. It should be noted that, we did not add any more data to our dataset.

While we do accept that given sufficient time and machine learning expertise, it may be possible to create adversarial samples that generalize across different models but it in effect makes evasion more expensive. If we can update the model frequently, adversaries will have to play catch-up every time.

Additionally, to improve the robustness of the models against adversarial attacks one could employ techniques like *min-max* (robust) optimization [56] ,where the classification loss is minimized while maximizing the acceptable perturbation one can apply to the image, or *randomized smoothing* [34, 51, 53] where provable (or certified) robust accuracy can be afforded. Such techniques have shown promising results in training robust models and are currently under active research [66, 78].

Two main criticisms with such techniques is the performance degradation in accuracy but also the costly optimization involved. Although the "inherent tension" between robustness and accuracy [72] is inevitable, the l_2 perturbations drive the network to focus on more perceptual features and not on imperceptual features that can be exploitable. The training time penalty though can be mitigated by adopting fast *min-max*-based adversarial robustness training algorithms like [67, 79]. Given the fast iteration time for fine-tuning our network, any such performance degradation should be within our iteration cycle quota. We leave thorough study of such mitigation techniques for future work.

Model	Size	FP	FN
Sentinel [22] Clone	256 MB	0/20	5/29
ResNet [42]	242 MB	0/20	21/39
PERCIVAL	1.76 MB	2/7	3/33

Figure 17: Tramer et al.'s [71] evaluation of various deep learning based perceptual ad blockers. The difference in the number of images used for evaluation stem from the kind of images the ad blocker is expecting.

7 Limitations

Dangling Text: By testing PERCIVAL integrated into Chromium, we noticed the following limitations. Many ads consist of multiple elements, which contain images and text information layered together. PERCIVAL is positioned in the rendering engine, and therefore it has access to one image at a time. This leads to situations where we effectively block the image, but the text is left dangling. Although this is rare, we can mitigate this by retraining the model with ad image frames containing just the text. Alternatively, a non-machine learning solution would be to memorize the DOM element that contains the blocked image and filter it out on consecutive page visitations. Although this might provide an unsatisfying experience to the user, we argue that it is of the benefit of the user to eventually have a good ad blocking experience, even if this is happening on a second page visit.

Small Images: Currently, images that are below 100×100 size skips PERCIVAL to reduce the processing time. This is a limitation which can be alleviated by deferring the classification and blocking of small images to a different thread, effectively blocking *asynchronously*. That way we make sure that we don't regress the performance significantly, while we make sure that consecutive requests will continue blocking small ads.

8 Related Work

Filter lists: Popular ad blockers like, Adblock Plus [1], uBlock Origin [26], and Ghostery [13] are using a set of rules, called filter-list, to block resources that match a predefined crowd-sourced list of regular expressions (from lists like EasyList and EasyPrivacy). On top of that, CSS rules are applied, to prevent DOM elements that are potential containers of ads. These filter-lists are crowd-sourced and updated frequently to adjust on the non-stationary nature of the online ads [70]. For example, EasyList, the most popular filter-list, has a history of 9 years and contains more than 60,000 rules [74]. However, filter-list based solutions enable a continuous cat-and-mouse game: their maintenance cannot scale efficiently, as they depend on the human-annotator and they do not generalize to "unseen" examples.

Perceptual Ad Blocking: Perceptual ad blocking is the idea of blocking ads based solely on their appearance; an example ad, highlighting some of the typical components. Storey et al. [70] uses the rendered image content to identify ads. More specifically, they use OCR and fuzzy image search techniques to identify *visual cues* such as ad disclosure markers or sponsored content links. Unlike PERCIVAL, this work assumes that the ad provider is complying with the legislation and is using visual cues like *AdChoices*.

Sentinel [65] proposes a solution based on convolutional neural networks (CNNs) to identify Facebook ads. This work is closer to our proposal; however, their model is not deployable in mobile devices or desktop computers because of its large size (>200MB). Also, we would like to mention the work of [28, 42, 77], where they use deep neural networks to identify the represented signifiers in the Ad images. This is a promising direction in semantic and perceptual ad blocking.

Adversarial attacks: In computer-vision, researchers have demonstrated attacks that can cause prediction errors by near-imperceptible perturbations of the input image. This poses risks in a wide range of applications on which computer vision is a critical component (e.g. autonomous cars, surveillance systems) [58–60]. Similar attacks have been demonstrated in speech to text [30], malware detection [39] and reinforcement-learning [41]. To defend from adversarial attacks, a portfolio of techniques has been proposed [33, 45, 46, 49, 50, 56], whether these solve this open research problem, remains to be seen.

9 Conclusion

With PERCIVAL, we illustrate that it is possible to devise models that block ads, while rendering images inside the browser. Our implementation shows a rendering time overhead of 4.55%, for Chromium and and 19.07%, for Brave browser, demonstrating the feasibility of deploying deep neural networks inside the critical path of the rendering engine of a browser. We show that our perceptual ad blocking model can replicate EasyList rules with an accuracy of 96.76%, making PERCIVAL a viable and complementary ad blocking layer. Finally, we demonstrate off the shelf language-agnostic detection due to the fact that our models do not depend on textual information and we show that PERCIVAL is a compelling blocking mechanism for first-party Facebook sponsored content, for which traditional filter based solutions are less effective.

Acknowledgements

We would like to thank the anonymous reviewers for their thoughtful comments and Brave Research team for providing valuable feedback during the project. Panagiotis Tigas is supported by the UK EPSRC CDT in Autonomous Intelligent Machines and Systems (grant reference EP/L015897/1). Zainul Abi Din is supported by a grant from Bouncer Technologies to UC Davis (grant reference A20-2169).

References

[1] Adblock Plus for Chrome support. https://adblockplus.org/.

[2] Annoying online ads do cost business. https://www.nngroup.com/articles/annoying-ads-cost-business/.

[3] Apple neutered ad blockers in Safari . https://www.zdnet.com/article/apple-neutered-ad-blockers-in-safari-but-unlike-chrome-users-didnt-say-a-thing/.

[4] Brave - Secure, Fast & Private Web Browser with Adblocker. https://brave.com/.

[5] Chromium Graphics. https://www.chromium.org/developers/design-documents/chromium-graphics.

[6] Coalition for better ads. https://www.betterads.org/research/.

[7] EasyList. https://easylist.to.

[8] ExpressVPN. https://www.expressvpn.com/.

[9] Facebook Ad placements. https://www.facebook.com/business/help/407108559393196.

[10] Facebook's Arms Race with Adblockers Continues to Escalate. https://motherboard.vice.com/en_us/article/7xydvx/facebooks-arms-race-with-adblockers-continues-to-escalate.

[11] False and deceptive display ads at yahoo's right media, 2009. http://www.benedelman.org/rightmedia-deception/#reg.

[12] Firefox's Enhanced Protection. https://blog.mozilla.org/blog/2019/09/03/todays-firefox-blocks-third-party-tracking-cookies-and-cryptomining-by-default/.

[13] Ghostery – Privacy Ad Blocker. https://www.ghostery.com/.

[14] Google Is Finally Making Chrome Extensions More Secure . https://www.wired.com/story/google-chrome-extensions-security-changes/.

[15] Implement hide-if-contains-snippet. https://issues.adblockplus.org/ticket/7088/.

[16] Introducing native ad blocking feature. . https://blogs.opera.com/desktop/2016/03/native-ad-blocking-feature-opera-for-computers/.

[17] iPhone users can block ads in Safari on iOS 9 . https://www.theverge.com/2015/6/11/8764437/iphone-adblock-safari-ios-9.

[18] More than half of local independent news sites are selling sponsored content. https://www.niemanlab.org/2016/06/more-than-half-of-local-independent-online-news-sites-are-now-selling-sponsored-content-survey/.

[19] Page Fair Ad Block Report. https://pagefair.com/blog/2017/adblockreport/.

[20] Puppeteer: Headless Chrome Node API. https://github.com/GoogleChrome/puppeteer.

[21] Selenium: Web Browser Automation. https://www.seleniumhq.org.

[22] Sentinel: The artificial intelligenence ad detector.

[23] Similar Web. https://www.similarweb.com/.

[24] Skia Graphics Library. https://skia.org/.

[25] The average web page is 3MB. https://speedcurve.com/blog/web-performance-page-bloat/.

[26] uBlock Origin. https://www.ublock.org/.

[27] Why people hate ads? https://www.vieodesign.com/blog/new-data-why-people-hate-ads/.

[28] Karuna Ahuja, Karan Sikka, Anirban Roy, and Ajay Divakaran. Understanding Visual Ads by Aligning Symbols and Objects using Co-Attention. 2018.

[29] Sruti Bhagavatula, Christopher Dunn, Chris Kanich, Minaxi Gupta, and Brian Ziebart. Leveraging Machine Learning to Improve Unwanted Resource Filtering. In *Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop - AISec '14*, 2014.

[30] Nicholas Carlini and David Wagner. Audio adversarial examples: Targeted attacks on speech-to-text. *Proceedings - 2018 IEEE Symposium on Security and Privacy Workshops, SPW 2018*, 2018.

[31] Claude Castelluccia, Mohamed-Ali Kaafar, and Minh-Dung Tran. Betrayed by your ads! reconstructing user profiles from targeted ads. In *Proceedings of the 12th International Conference on Privacy Enhancing Technologies*, PETS'12, page 1–17, Berlin, Heidelberg, 2012. Springer-Verlag.

[32] Quan Chen, Peter Snyder, Ben Livshits, and Alexandros Kapravelos. Improving web content blocking with event-loop-turn granularity javascript signatures. *arXiv*, pages arXiv–2005, 2020.

[33] Steven Chen, Nicholas Carlini, and David A. Wagner. Stateful detection of black-box adversarial attacks. *CoRR*, abs/1907.05587, 2019.

[34] Jeremy M Cohen, Elan Rosenfeld, and J Zico Kolter. Certified adversarial robustness via randomized smoothing. In *Proceedings of the 36th International Conference on Machine Learning*, Proceedings of Machine Learning Research, pages 1310–1320. PMLR, 2019.

[35] Steven Englehardt and Arvind Narayanan. Online tracking: A 1-million-site measurement and analysis. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, New York, NY, USA, 2016. ACM.

[36] G. Storey, D. Reisman, J. Mayer and A. Narayanan. Perceptual Ad Highlighter. https://chrome.google.com/webstore/detail/perceptual-ad-highlighter/mahgiflleahghaapkboihnbhdplhnchp, 2017.

[37] Kiran Garimella, Orestis Kostakis, and Michael Mathioudakis. Ad-blocking: A Study on Performance, Privacy and Counter-measures. In *Proceedings ofWebSci '17*, Troy, NY, USA, 2017. WebSci '17.

[38] Daniel G. Goldstein, R. Preston McAfee, and Siddharth Suri. The cost of annoying ads. In *WWW '13*, 2013.

[39] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. Adversarial examples for malware detection. In *European Symposium on Research in Computer Security*. Springer, 2017.

[40] David Gugelmann, Markus Happe, Bernhard Ager, and Vincent Lenders. An Automated Approach for Complementing Ad Blockers' Blacklists. In *Proceedings on Privacy Enhancing Technologies*, volume 2015, 2015.

[41] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*, 2017.

[42] Zaeem Hussain, Christopher Thomas, Mingda Zhang, Zuha Agha, Xiaozhong Zhang, Nathan Ong, Keren Ye, and Adriana Kovashka. Automatic understanding of image and video advertisements. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-January(14):1100–1110, 2017.

[43] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. 2016.

[44] Umar Iqbal, Zubair Shafiq, Peter Snyder, Shitong Zhu, Zhiyun Qian, and Benjamin Livshits. Adgraph: A machine learning approach to automatic and effective adblocking. *CoRR*, abs/1805.09155, 2018.

[45] Harini Kannan, Alexey Kurakin, and Ian Goodfellow. Adversarial Logit Pairing. *arXiv preprint arXiv:1803.06373*, 2018.

[46] J Zico Kolter and Eric Wong. Provable defenses against adversarial examples via the convex outer adversarial polytope. *arXiv preprint arXiv:1711.00851*, 1(2), 2017.

[47] Georgios Kontaxis and Monica Chew. Tracking protection in firefox for privacy and performance. *ArXiv*, abs/1506.04104, 2015.

[48] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, USA, 2012. Curran Associates Inc.

[49] Alex Kurakin, Dan Boneh, Florian Tramèr, Ian Goodfellow, Nicolas Papernot, and Patrick McDaniel. Ensemble Adversarial Training: Attacks and Defenses. 2018.

[50] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

[51] Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. Certified robustness to adversarial examples with differential privacy. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 656–672. IEEE, 2019.

[52] Adam Lerner, Anna Kornfeld Simpson, Tadayoshi Kohno, and Franziska Roesner. Internet jones and the raiders of the lost trackers: An archaeological study of web tracking from 1996 to 2016. In *25th USENIX Security Symposium (USENIX Security 16)*, Austin, TX, August 2016. USENIX Association.

[53] Bai Li, Changyou Chen, Wenlin Wang, and Lawrence Carin. Certified adversarial robustness with additive noise. In *Advances in Neural Information Processing Systems*, pages 9459–9469, 2019.

[54] Zhou Li, Kehuan Zhang, Yinglian Xie, Fang Yu, and XiaoFeng Wang. Knowing your enemy: Understanding and detecting malicious web advertising. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 674–686, New York, NY, USA, 2012. ACM.

[55] Timothy Libert. Exposing the hidden web: An analysis of third-party http requests on 1 million websites. 11 2015.

[56] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

[57] G. Merzdovnik, M. Huber, D. Buhov, N. Nikiforakis, S. Neuner, M. Schmiedecker, and E. Weippl. Block me if you can: A large-scale study of tracker-blocking tools. In *2017 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 319–333, 2017.

[58] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples. 2016.

[59] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical Black-Box Attacks against Machine Learning. 2016.

[60] Nicolas Papernot, Patrick Mcdaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. *Proceedings - 2016 IEEE European Symposium on Security and Privacy, EURO S and P 2016*, 2016.

[61] Enric Pujol, Tu Berlin, Oliver Hohlfeld, Anja Feldmann, and Tu Berlin. Annoyed users: Ads and ad-block usage in the wild.

[62] Sam Tolomei. Shrinking APKs, growing installs. https://medium.com/googleplaydev/shrinking-apks-growing-installs-5d3fcba23ce2.

[63] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018.

[64] Ramprasaath R. Selvaraju, Abhishek Das, Ramakrishna Vedantam, Michael Cogswell, Devi Parikh, and Dhruv Batra. Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization. *CoRR*, abs/1610.02391, 2016.

[65] Adblock Sentinel. Adblock Plus, Sentinel, 2018.

[66] Ali Shafahi, Mahyar Najibi, Mohammad Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! In *Advances in Neural Information Processing Systems*, pages 3353–3364, 2019.

[67] Ali Shafahi, Mahyar Najibi, Mohammad Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 3358–3369. Curran Associates, Inc., 2019.

[68] R. Shao, V. Rastogi, Y. Chen, X. Pan, G. Guo, S. Zou, and R. Riley. Understanding in-app ads and detecting hidden attacks through the mobile app-web interface. *IEEE Transactions on Mobile Computing*, 17(11):2675–2688, 2018.

[69] Anastasia Shuba and Athina Markopoulou. NoMoATS: Towards Automatic Detection of Mobile Tracking. *Proceedings on Privacy Enhancing Technologies*, 2020(2), 2020.

[70] Grant Storey, Dillon Reisman, Jonathan Mayer, and Arvind Narayanan. The Future of Ad Blocking: An Analytical Framework and New Techniques. 2017.

[71] Florian Tramèr, Pascal Dupré, Gili Rusak, Giancarlo Pellegrino, and Dan Boneh. Ad-versarial: Defeating perceptual ad-blocking. *CoRR*, abs/1811.03194, 2018.

[72] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. In *International Conference on Learning Representations*, 2019.

[73] Narseo Vallina-Rodriguez, Jay Shah, Alessandro Finamore, Yan Grunenberger, Konstantina Papagiannaki, Hamed Haddadi, and Jon Crowcroft. Breaking for commercials: Characterizing mobile advertising. In *Proceedings of the 2012 Internet Measurement Conference*, IMC '12, page 343–356, New York, NY, USA, 2012. Association for Computing Machinery.

[74] Antoine Vastel, Peter Snyder, and Benjamin Livshits. Who Filters the Filters: Understanding the Growth, Usefulness and Efficiency of Crowdsourced Ad Blocking. *arXiv preprint arXiv:1810.09160*, 2018.

[75] Qianru Wu, Qixu Liu, Yuqing Zhang, Peng Liu, and Guanxing Wen. A machine learning approach for detecting third-party trackers on the web. In Sokratis Katsikas, Catherine Meadows, Ioannis Askoxylakis, and Sotiris Ioannidis, editors, *Computer Security - 21st European Symposium on Research in Computer Security, ESORICS 2016, Proceedings*, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), pages 238–258, Germany, January 2016. Springer Verlag. 21st European Symposium on Research in Computer Security, ESORICS 2016 ; Conference date: 26-09-2016 Through 30-09-2016.

[76] Xinyu Xing, Wei Meng, Byoungyoung Lee, Udi Weinsberg, Anmol Sheth, Roberto Perdisci, and Wenke Lee. Understanding malvertising through ad-injecting browser extensions. In *Proceedings of the 24th international conference on world wide web*. International World Wide Web Conferences Steering Committee, 2015.

[77] Keren Ye and Adriana Kovashka. ADVISE: Symbolism and external knowledge for decoding advertisements. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11219 LNCS, 2018.

[78] Dinghuai Zhang, Tianyuan Zhang, Yiping Lu, Zhanxing Zhu, and Bin Dong. You only propagate once: Accelerating adversarial training via maximal principle. In *Advances in Neural Information Processing Systems*, pages 227–238, 2019.

[79] Dinghuai Zhang, Tianyuan Zhang, Yiping Lu, Zhanxing Zhu, and Bin Dong. You only propagate once: Accelerating adversarial training via maximal principle. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 227–238. Curran Associates, Inc., 2019.